

Migration Specialties International, Inc.

217 West 2nd Street, Florence, CO 81226-1403

719-371-1711, Fax: 888-854-3417

E-mail: Info@MigrationSpecialties.com

www.MigrationSpecialties.com



Bruce Claremont, July 2005

Using OpenVMS to Meet a Sarbanes-Oxley Mandate

Part 1: The Project

Introduction.....	2
Sarbanes-Oxley Project Overview	2
The Task.....	2
Preparation	2
Project Plan.....	3
Implementing the Plan	4
Procedure Identification.....	4
Procedure Updates	6
Controlling Future Development	7
Testing.....	7
Batch Queues	7
Log File Retention	7
Documentation.....	8
Phase One Conclusion	9
Phase Two.....	9

INTRODUCTION

Migration Specialties has maintained a set of OpenVMS applications for a large multi-national corporation for many years. This particular set of applications is being retired, but our expertise with OpenVMS led to our being asked to participate in a short, intense project to bring their remaining OpenVMS applications into compliance with the Sarbanes-Oxley Act. In brief, the Sarbanes-Oxley Act mandates that large companies operating in the United States attest to the financial and IT controls they have in place in their annual reports to stock holders and the federal government.

This article covers our efforts in two parts. The first part discusses how we approached the project, set goals, and achieved them. The second part, which will appear as a second article, covers in some detail how we took advantage of OpenVMS and DCL to attain our goals. Part 2 provides examples of some of the DCL code we used to implement our compliance strategies. The article highlights how well the OpenVMS operating system coupled with good coding practices lends itself to Sarbanes-Oxley compliance.

SARBANES-OXLEY PROJECT OVERVIEW

The Task

A mandate had been handed down by corporate IT that procedures be put in place to comply with aspects of the Sarbanes-Oxley Act. In this instance, corporate IT had decided that part of this effort would involve development of application error recovery procedures and retention of all batch process logs. An internal audit was taking place at the time and Sarbanes-Oxley compliance was being checked in preparation for a formal audit later in the year. Our task was to take this general assignment and apply it to the OpenVMS applications supported by the local IT department. We had five weeks to meet the initial goal of passing the internal audit.

On the surface, this task sounds pretty simple. Batch log files can be retained as long as one cares to by not deleting them. Error capture, reporting, and recovery should be a documented process in an organization of any size. Eliminate a few purges, update a few operation manuals, and we'd be all set.

In reality, few companies allow their IT organizations the latitude and resources to be that well organized or prepared. Typical of many sites, OpenVMS applications in use were an eclectic mix of old and new that had passed through the hands of many programmers. Some batch procedures logged; some didn't. Error recovery was informal and relied upon the knowledge of a few key people. Some of these people had moved into management positions over time, but were still dragged back into application code when problems occurred. It was very much an environment of fix things when they break; otherwise, concentrate on the work at hand.

Preparation

The first order of business was to assess the magnitude of the task we had been assigned and see what was feasible in the time frame allotted. One employee and I were assigned to the task full time. We had the services of two managers, another employee, and an outside consultant as their schedules allowed; all were familiar with the applications in use.

We made a quick first pass at scaling the project by listing all DCL command procedures on the OpenVMS systems. System, application, and user directories were well separated, so we could easily identify application specific procedures, which were our primary target. Our initial analysis returned the following results:

Table 1: Overall Procedure Count

Location	Static Procedures	Dynamic Procedures	Total
Site A	1,602	312	1,914
Site B	1,747	446	2,193
Shared	53	0	53
Totals	3,402	758	4,160

Static Procedure: Permanent procedures that rarely change.

Dynamic Procedure: Procedures that are dynamically generated by other procedures or programs.

It was quickly apparent that it would be unrealistic to wade through over 4,000 procedures, document formal error handling processes, and test everything in the time available. Our next step was to devise a realistic plan to meet as many management goals as possible.

Project Plan

We quickly developed a short project plan that accomplished two important goals:

1. Defined what we would accomplish.
2. Limited the scope of our project.

Our available resources and the volume of work required meant we couldn't deliver everything expected in the time frame allowed. We tackled this limitation by breaking the project into two phases. Phase one would deliver critical elements and provide minimal compliance with Sarbanes-Oxley. Phase two would deliver the remaining elements and provide full Sarbanes-Oxley compliance.

In phase one we planned to:

- Identify all active DCL command procedures.
- Identify a subset of active DCL procedures deemed critical to ongoing operations.
- Standardize logging and error trapping in all critical DCL procedures.
- Retain logs from all batch procedures for 12 months.
- Provide automated notification of batch job errors via e-mail.
- Implement manual error report and follow-up procedures for all OpenVMS batch procedure errors.
- Generate documentation outlines for implemented processes and procedures.

Phase two would cover:

- Standardization of logging and error trapping in all remaining active DCL procedures.
- Identification of departments that "owned" critical procedures and that are responsible for responding to error reports.
- Automation of log retention.
- Automation of error reporting.
- Full documentation of all processes and procedures.

Note that we proposed to standardize *all* active DCL procedures, not just batch procedures. Over the years, Migration Specialties has taken advantage of OpenVMS features to develop an effective methodology for building standardized DCL procedures irrespective of whether they are used interactively or as batch processes. This methodology greatly simplifies procedure development and maintenance and the client agreed to deploy it as part of their Sarbanes-Oxley compliance effort. Examples of the DCL code we deployed as part of this project are discussed in greater detail in the second part of this article.

Our formal project plan was short and concise, covering less than two pages. It justified our decisions based on resources at hand and the time frame allowed. Management quickly signed off on the plan and authorized us to proceed.

Implementing the Plan

Our project plan had four main components:

- Identify active DCL procedures.
- Modify active DCL procedures.
- Test modified procedures.
- Document error handling processes.

In phase one, we would identify all active procedures, but only modify a subset. Our documentation would be preliminary, serving to outline the full error tracking and recovery process. Phase two would see to full implementation of our procedure changes and documentation efforts.

Procedure Identification

Our first order of business was to cull through all of the application DCL procedures on the systems and identify those that were still in use. Over time, most application directories tend to accumulate obsolete procedures, test procedures, and temporary procedures. We wanted to eliminate this detritus before starting the review and modification process. We accomplished this with a combination of methods:

- We used the OpenVMS SEARCH command to locate procedure calls via the @ symbol and SUBMIT command in DCL procedure and program source code.
- Many of the applications used text file driven menu systems, so we scanned the menu files for procedure names.
- We queried system operations staff and users, asking them to list any procedures they ran manually.

Via the OpenVMS Accounting Utility, we were able to positively identify procedures that had been run over the past four months. We correlated this information with the data we had gathered in the previous three steps, building a spreadsheet that listed the procedures and their status. Procedures appearing in the Accounting logs were automatically deemed active. Our spreadsheet looked like this:

Table 2: Procedure Analysis Spreadsheet

Procedure Name	Batch Queue	User ID	Runs	Status
MFG01	SYS\$BATCH	JACKSON	109	Active
PAYRATE	SYS\$BATCH	LONGS	16	Active
SHOP_UPDATE	SINGLE\$BATCH	MEGAN	4	Unknown

A set of Microsoft Excel add-ons available through DigDB.com proved invaluable during this process. The DigDB add-ons, coupled with Excel's filtering capabilities, allowed us to merge lists, remove duplicates, and quickly distill our data down to the essentials.

We circulated the refined spreadsheet among the operations staff with the most application experience and asked them to do the following:

- Verify that procedures marked active were still in use.
- Review the procedures marked as unknown. If they were familiar with the procedure, they could mark it as follows:
 - **Critical:** Procedure is critical to ongoing operations.
 - **Active:** Procedure is still in use.
 - **Adhoc:** Procedure is run on an adhoc basis and is not considered part of the production environment.
 - **Remove:** The procedure is obsolete and can be removed from the production environment.

The end result of all of this analysis was a list of critical operational procedures. As Table 3 illustrates, our procedure count was considerably reduced. The analysis also provided a list of obsolete procedures. We took advantage of this information and archived all obsolete and unknown procedures, effectively removing them from the production stream. Should one of these procedures prove to be needed in the future, it was still available, but would need to be placed back in the production area. For our purposes, these procedures were not considered in our review and update process.

Table 3: Critical Procedure Count

Location	Static Procedures	Dynamic Procedures	Total
Site A	92	47	139
Site B	104	0	104
Shared	56	0	56
Totals	252	47	299

Procedure Updates

The overall goal of our DCL procedure modifications was to automate and enforce strict error trapping and reporting mechanisms. Our strategy was to implement a set of standardized procedure initialization and termination routines such that all OpenVMS procedures followed the same set of rules when they terminate. This allowed us to control how terminations are performed and capture the information necessary to comply with Sarbanes-Oxley rules.

Our plan to bring the DCL procedures into compliance involved the following steps:

- Agree upon the initialization and termination code to be added to each DCL command procedure. The code was supply by Migration Specialties and modified to fit the client's requirements.
- Develop automated tools to examine all procedures and identify code that was out of compliance.
- Develop automated tools to apply the standardized code to all existing active procedures.
- Manually review procedures identified as out of compliance and correct them.

We enforced termination control by capturing all errors and user aborts via the ON ERROR and ON CONTROL_Y commands. Most of the procedures we worked on did not have any error trap mechanisms, but some did. To ensure our controls were not superceded within the body of a procedure, we modified existing error trap mechanisms to exit through our standard termination routine.

Our automated error notification scheme involved construction of an e-mail message containing the procedure name, log file name, user name, and a time stamp. In phase one, the message was routed to one of two facility IT managers. The managers were responsible for forwarding the message to the appropriate party for investigation and correction. Under phase two, error message routing was to be refined to target personnel more directly involved with the application in use.

We built our tools using DCL. This allowed us to rapidly develop and test our applications. Part 2 contains details on the tools and code used in this endeavor. To test our applications, we concentrated our initial efforts on the 56 shared procedures. We ran them through our compliance tool, analyzed the results, inserted our standardized initialization and termination routines, and corrected non-compliance issues. We then tested the applications. We used the results to fine tune our automated tools and update process. After three passes on this initial set of code, we were ready to take on the remaining critical procedures. Our trial runs established a procedure review and update rate for an experienced DCL programmer of 10 – 15 procedures an hour. With the tools and a defined process in place, it was simply a matter of grinding through the remaining procedures and bringing them into compliance.

The DCL SEARCH command was invaluable throughout this process. It allowed us to quickly ascertain whether DCL statements that were out of compliance were consistently coded. If so, they could be quickly corrected with a mass update using the EDT editor and the SUBSTITUTION command.

Where manual review and updates were required, TPU and its ability to memorize and repeat key sequences was very helpful. Memorized commands allowed files to be quickly and consistently reviewed and updated.

Dynamic procedures were a bit more challenging. Since they were generated within another procedure or program, they required manual review and update. There were not enough of them to justify building a more sophisticated analysis tool, nor was there time to do so. Fortunately, most of the generated procedures were very simple and were brought into compliance through the manual insertion of our standard initialization and termination modules.

Our careful analysis and automation paid off. We successfully updated and tested all of the critical procedures identified in phase one in the time allotted. We were also able to make initial analysis and standardization passes on the remaining procedures, preparing them for review and testing in phase two.

Controlling Future Development

To ensure our Sarbanes-Oxley modifications would propagate to new procedures, we developed and documented a procedure template. The template is discussed in detail in Part 2. The template was incorporated into the company coding standards. Its use is required in all future development.

Testing

Our client had a test bed and formal testing processes in place, so testing our changes was reasonably routine. There were some surprises when our error traps revealed that some procedures were not executing correctly and never had. The procedures had either contained SET NOON commands and bypassed internal errors or had simply exited and never been caught. Fortunately, none of these problems was severe enough to delay our delivery.

Batch Queues

We instigated error trapping on all 74 production batch queues using the SET QUEUE /RETAIN=ERROR. We then modified a Migration Specialties supplied utility to scan the batch queues on a scheduled basis and report any jobs that had terminated with errors. This process provided redundancy, ensuring that if a failed batch procedure did not send out a notification immediately, one would still be sent. Furthermore, notifications would continue to be sent until the failure was examined and the retained entry removed from the batch queue.

As with the procedure error traps, errors found in the batch queues were reported via e-mail to the two facility managers. Phase two plans called for more refined error message routing to target personnel more directly involved with the application in use.

Unlike our initial revisions to a subset of active DCL procedures, the batch queue change impacted all batch jobs running on the system. Instigation of error trapping and reporting at the batch queue level immediately revealed several jobs that were terminating with errors. Some of these jobs were obsolete and could be eliminated. Others proved to be jobs that were failing without being caught, causing some consternation amongst the IT staff and users.

Log File Retention

Log file retention was straightforward. We estimated the space necessary to retain one year's worth of log files, then allocated the necessary disk resources and set up designated areas in which each application could deposit its log files. We recognized a potential issue with version number wrap when version numbers reached 32,767 and made plans to address it in phase two. We also initiated plans to archive log files to tape after three months to conserve disk space.

As part of our DCL procedure review and update process, we removed all commands that purged or deleted log files. We verified that all directories designated to retain log files did not have version limits in place. We reviewed all batch submission command lines and implemented a standard submission command string using the SUBMIT /LOG=filename.LOG /KEEP /NOPRINT command.

Documentation

We completed formal documentation of our procedure development template, DCL standardization rules, and batch queue monitoring utility during phase one. We outlined our phase one operational procedures, audit events, and batch log retention plan. We also outlined these items for phase two.

Phase one operational procedures were defined as follows:

1. Batch process errors were automatically trapped and reported via e-mail by the affected batch procedure.
2. Error reports were automatically routed to one of two designated facility IT managers. These managers were responsible for seeing that the impacted user was notified of the problem and that remedial action was taken.
3. In conjunction with the batch job generated error notification, the facility IT managers also received automatic notification of the problem via the batch queue error monitoring process. They were responsible for monitoring notifications and responses to ensure that errors were being dealt with in a timely manner.
4. All batch logs were retained for a minimum of one year, ensuring an audit could review past batch activity over a 12-month time frame.

Successful implementation of these objectives was deemed to be minimally compliant with Sarbanes-Oxley requirements.

Phase two operational procedures were outlined as follows:

1. Batch process errors would be automatically trapped and reported via e-mail by the affected batch procedure.
2. Procedure generated error notification would be automatically routed to the user that initiated the batch process. The user was responsible for forwarding the report to the IT department with a notation concerning the problem severity. A special e-mail form was designed for this purpose.
3. Upon notification, the IT department would be required to provide a proposed resolution response within 24 hours. The problem report would be submitted to normal IT channels for resolution, testing, and roll-out. The IT department would be required to issue a final notification when the problem had been officially resolved.
4. In conjunction with the user error notification, the impacted department head and local IT management would receive automatic notification of the problem via the batch queue error monitoring process. They were responsible for monitoring notifications and responses to ensure that errors were being dealt with in a timely manner. Jobs retained on error could not be removed from the batch queues until the IT department had received notification from the user of the problem and their assessment of its severity.
5. All batch logs were retained for a minimum of one year, ensuring an audit could review past batch activity over a 12-month time frame.

We on the implementation team thought the phase two requirements were a bit onerous and clumsy. However, this is what was handed down by a team of corporate attorneys, auditors, and IT personnel tasked with defining Sarbanes-Oxley requirements. It represented their interpretation of the Sarbanes-Oxley rules and regulations. It was our job to implement their vision.

Phase One Conclusion

Phase one was completed successfully. We even exceeded our goals by a little bit. Careful planning and analysis up front yielded lots of dividends as we worked through the phase one tasks. Many man-hours and a fair bit of overtime were devoted to our effort, but at no point did the project ever feel desperate or uncontrolled. On the contrary, as the project moved forward we built a nice sense of momentum and felt we were well positioned to tackle phase two.

Phase Two

Phase two didn't happen. As we completed phase one, the local IT department was hit by a round of layoffs and outsourcing that severely disrupted operations. Phase two was put on indefinite hold.

***About the author:** Mr. Bruce Claremont has a degree in Computer Science and has been working with OpenVMS since 1983. Bruce has extensive programming, project management, and system management experience. He has worked all sides of the fence, as a customer, software engineer, system manager, delivery specialist, project manager, and business owner. He founded Migration Specialties in 1992 and continues to deliver OpenVMS, software migration, and hardware emulation services. Bruce would like to help you achieve similar success in porting and supporting your applications. More information about Migration Specialties products and services can be found at www.MigrationSpecialties.com.*